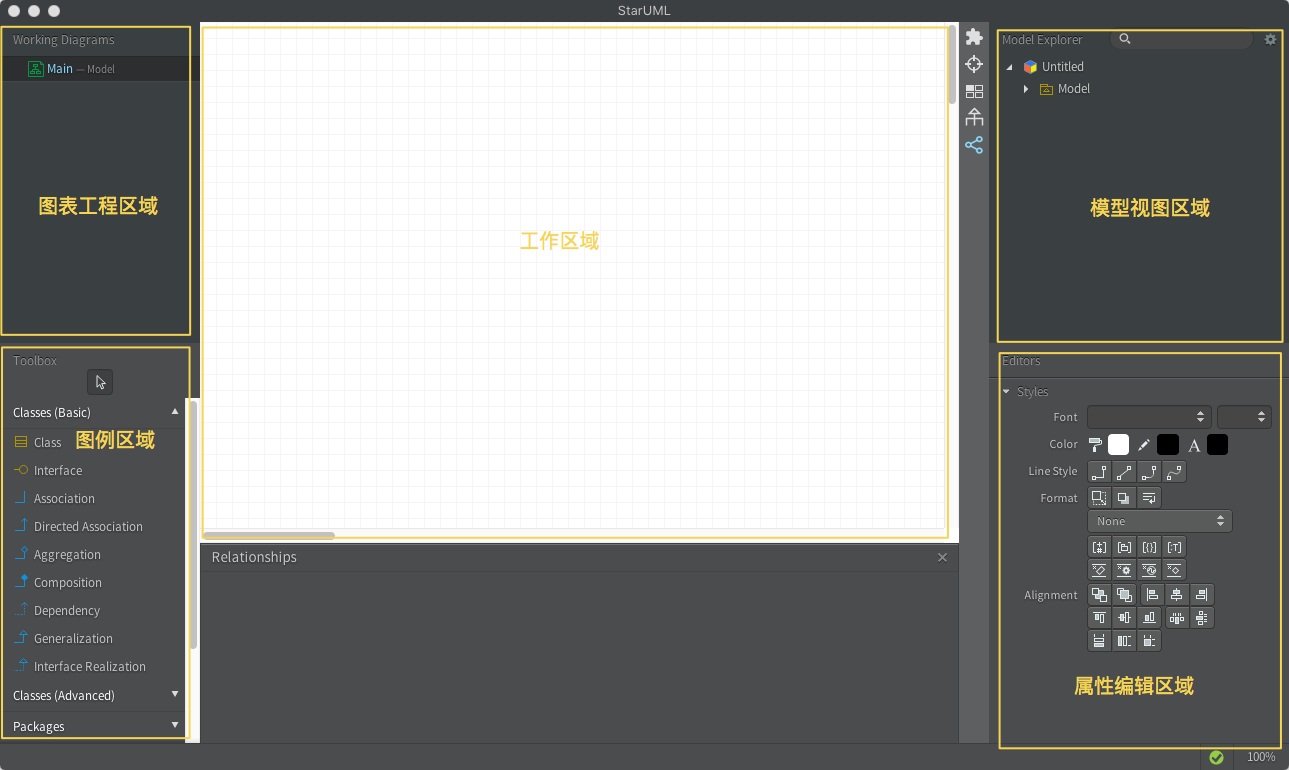
# StarUML3

<https://www.jianshu.com/p/abe2df1b96cf>

## 软件介绍



图表工程区域： 所有新建的图表都是展示在此处，可通过Model-->Add Diagram新建不同的图表。

图例区域：展示绘制图表的元素

工作区域： 绘制图表的区域

模型视图区域：树状结构展示图表中各个元素的关系

属性编辑区域：编辑图表相关属性，如文字颜色、大小，线的粗细等；以及相关元素的相关属性也在这边设置。

## 新建工程

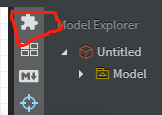
启动即新建，也可File-New

## 添加类图

Model-Add Diagram-Class Diagram

## 安装插件

java插件（生成与逆向）、参数自动生成set get插件（演示下） 等。



### 生成stub code

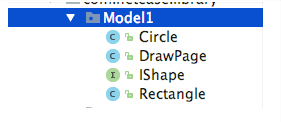
StarUML能够自动生成Java的“stub code” 的工具。也可以做JAVA逆向工程，以产生相应的UML图表。

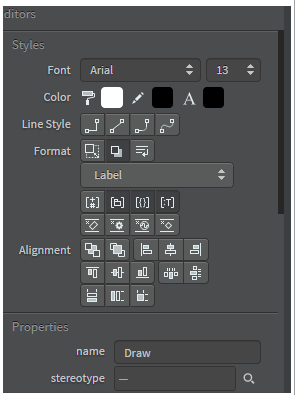
选择“Tools — Java — Generate Code”。

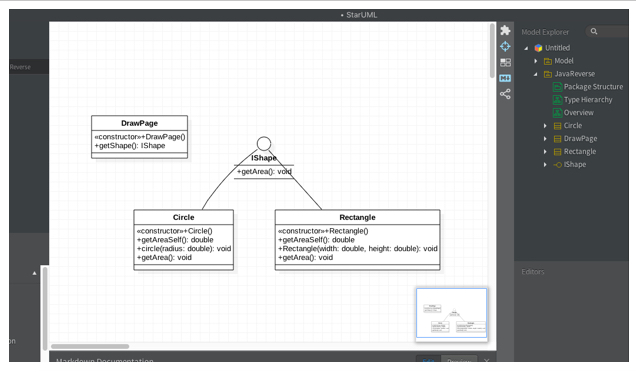
### 逆向工程

StarUML还可以从现有的Java代码创建一个类图，这被称为“reverse engineering”。当你想从现有的代码生成图表，或者你修改了SU生成的代码，并且想在图表中反应出来的时候，逆向工程功能就非常有用了。

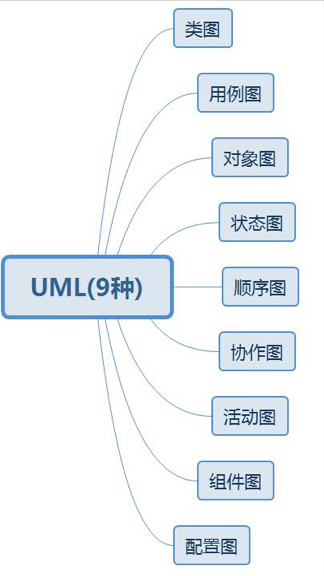
到主菜单栏中选择“Tools — Java — Reverse Engineer…”，可以将现有的代码逆向工程。







## 基本使用

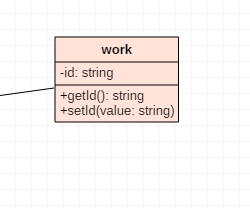


今天重点分享下 类图、状态图、活动图、顺序图、用例图 ， 五种最常用图的绘制。

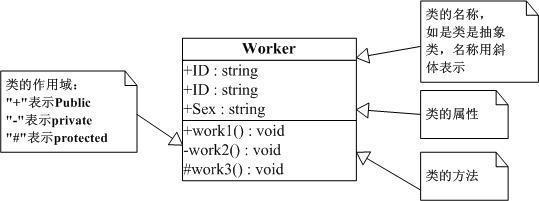
### 类图

使用类来描述系统的静态结构，类图包含类和它们之间的关系，它描述系统内所声明的类，但它没有描述系统运行时类的行为。

在UML类图中，类一般由三部分组成：

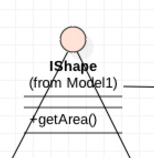


类是对现实世界中一组具有相同特征的物体的抽象。

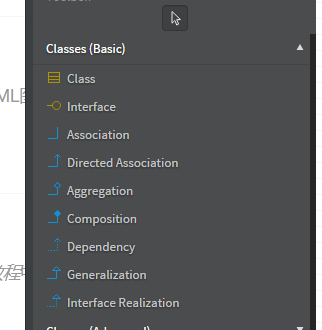


**接口（Interface）**

是一种特殊的类，具有类的结构但不可被实例化，只可以被实现（继承）。在UML中，接口使用一个带有名称的小圆圈来进行表示。



#### 类之间的关系



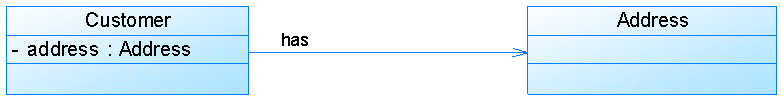
##### 关联关系(Association)

类与类之间最常用的一种关系，它是一种结构化关系，用于表示一个类对象与另一个类对象之间有联系。

在 UML 类图中，用实线连接有关联的对象所对应的类。在实现关联关系时，通常将一个类的对象作为另一个类的属性。

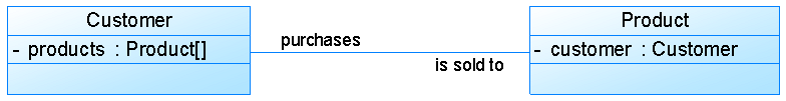
**单向关联**

类的关联关系可以是单向的，单向关联用带箭头的实线表示。

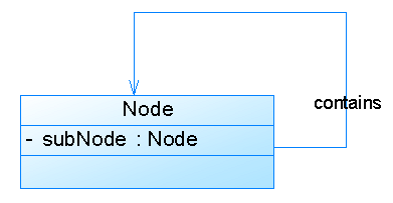


Ctril + L 弄直

**双向关联**



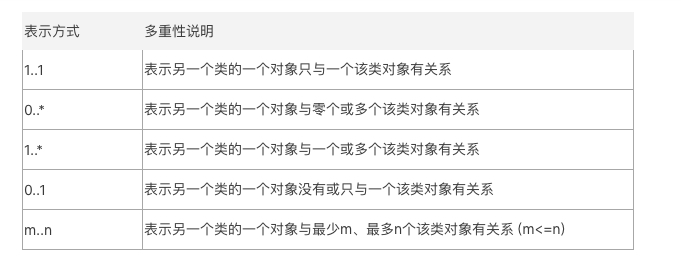
**自关联**

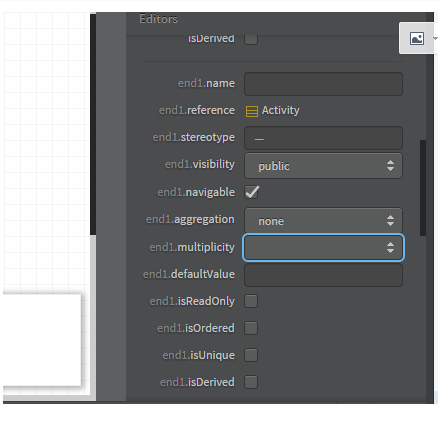


**重数性关联**

重数性关联关系又称为多重性关联关系(Multiplicity)，表示一个类的对象与另一个类的对象连接的个数。在 UML 中多重性关系可以直接在关联直线上增加一个数字表示与之对应的另一个类的对象的个数。







##### 聚合关系(Aggregation)

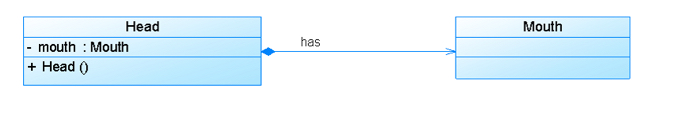
表示一个整体与部分的关系。在聚合关系中，成员类是整体类的一部分，即成员对象是整体对象的一部分，成员对象可以脱离整体对象独立存在。 在 UML 中，聚合关系用带空心菱形的直线表示。





##### 组合关系(Composition)

组合关系也表示类之间整体和部分的关系，但是组合关系中部分和整体具有统一的生存期。一旦整体对象不存在，部分对象也将不存在，部分对象与整体对象之间具有同生共死的关系。在UML中，组合关系用带实心菱形的直线表示。

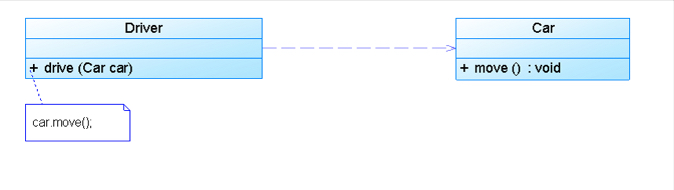


##### 依赖关系(Dependency)

依赖关系是一种使用关系，特定事物的改变有可能会影响到使用该事物的其他事物，在需要表示一个事物使用另一个事物时使用依赖关系。

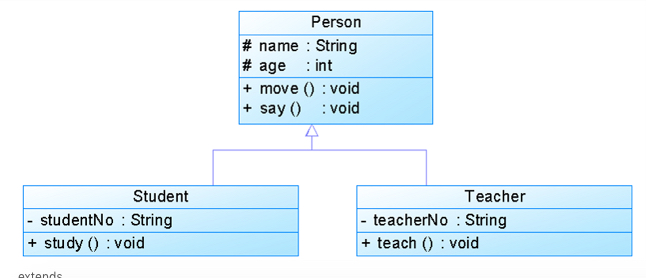
大多数情况下，依赖关系体现在某个类的方法使用另一个类的对象作为参数。

在UML中，依赖关系用带箭头的虚线表示，由依赖的一方指向被依赖的一方。



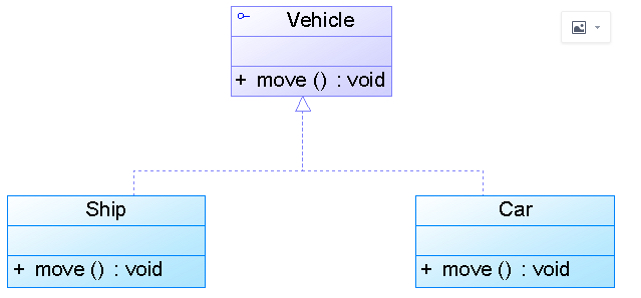
##### 泛化关系(Generalization)

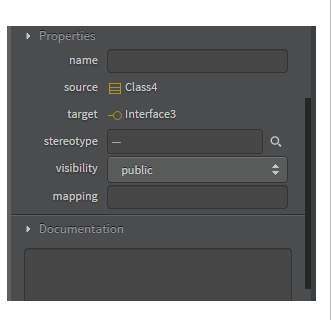
也就是继承关系，也称为“is-a-kind-of”关系，泛化关系用于描述父类与子类之间的关系，父类又称作基类或超类，子类又称作派生类。在 UML 中，泛化关系用带空心三角形的直线来表示。



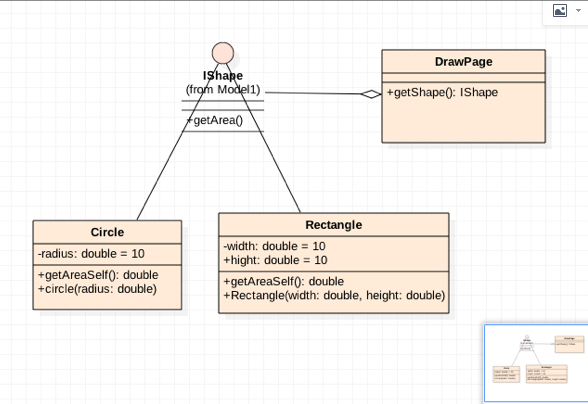
##### 实现关系(Realization)

实现关系是类实现了接口，类中的操作实现了接口中所声明的操作。在UML中，类与接口之间的实现关系用带空心三角形的虚线来表示。



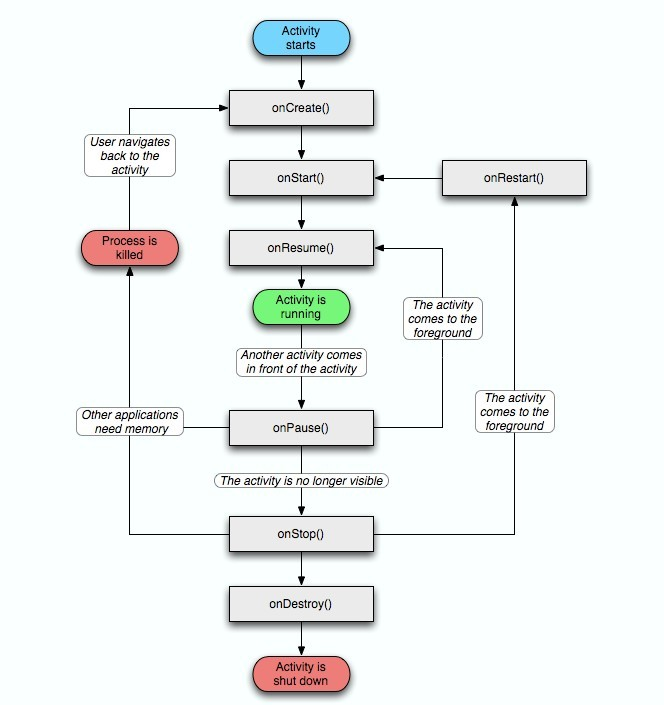


示例：

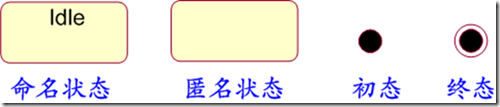


### 状态图（Statechart Diagram）

状态是指在对象生命周期中满足某些条件、执行某些活动或等待某些事件的一个条件和状况 （比如说Activity生命周期状态，不怎么标准的状态图）



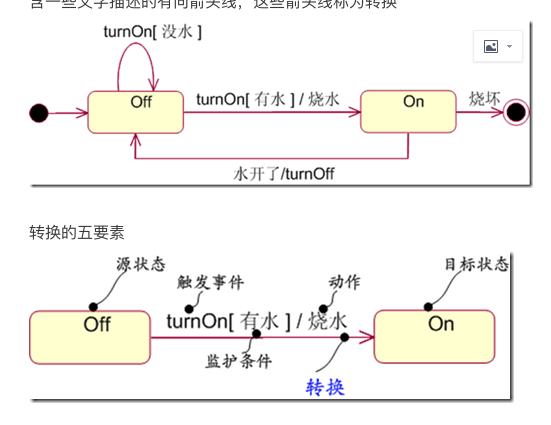
一个状态通常包括名称、进入/退出活动、内部转换、子状态和延迟事件 等五个部分组成





#### 最简单的状态图

最为核心的元素无外乎是两个：一个是用圆角矩形表示的状态 （初态和终态例外）；另一个则是在状态之间的、包含一些文字描述的有向箭头线，这些箭头线称为转换



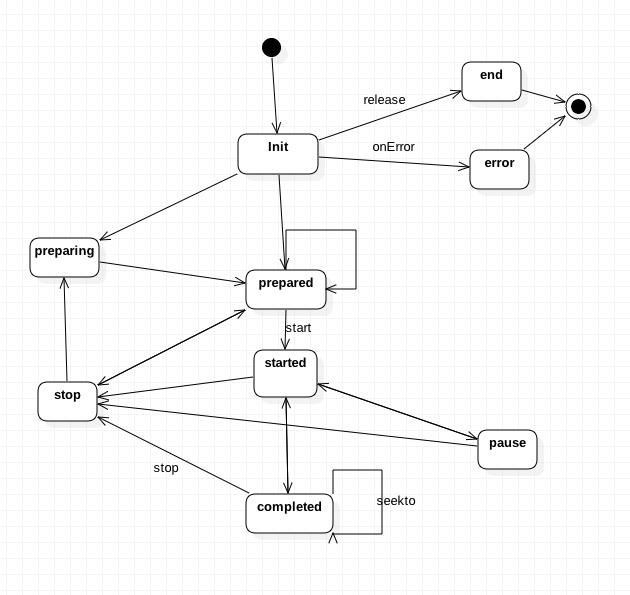
源状态：即受转换影响的状态

目标状态：当转换完成后对象的状态

触发事件：用来为转换定义一个事件，包括调用、改变、信号、时间四类事件

监护条件：布尔表达式，决定是否激活转换、

动作：转换激活时的操作



#### 带有复杂转换的状态图

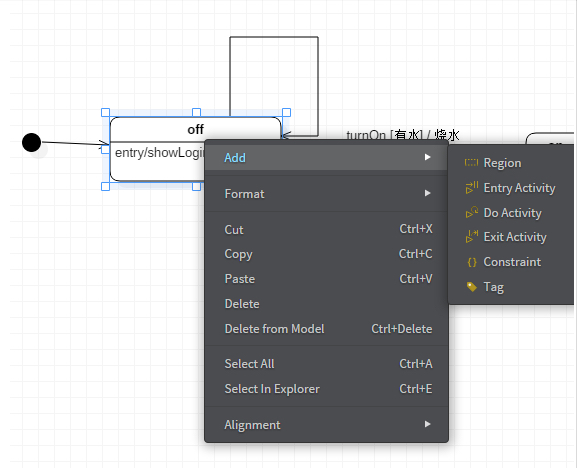
只有动作描述，进入和退出和操作方法写在了里面

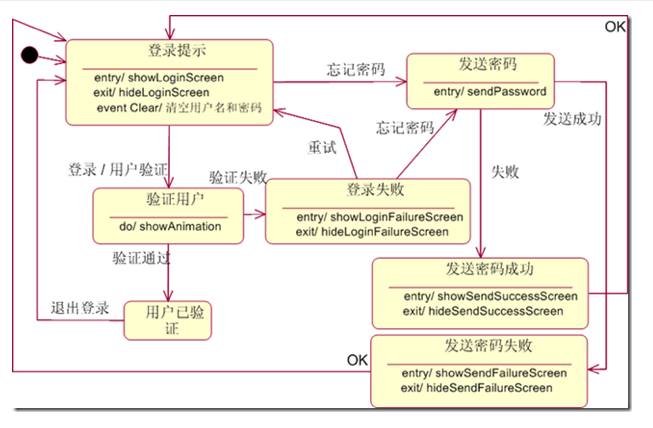
进入和退出转换 ：当进入一个状态时，执行某个动作；或当退出某个状态时，执行什么动作。 这时就可以使用进入和退出转换 来表示

内部转换 ：用来处理一些不离开该状态的事件

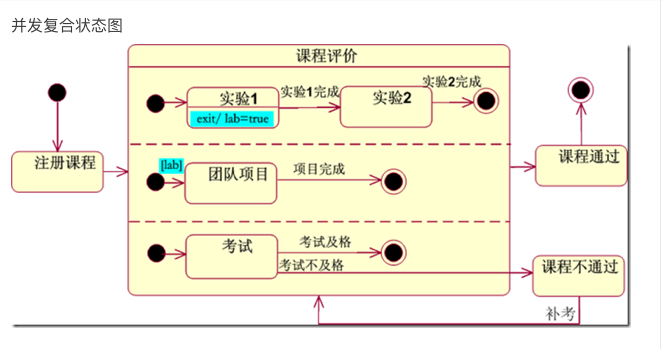






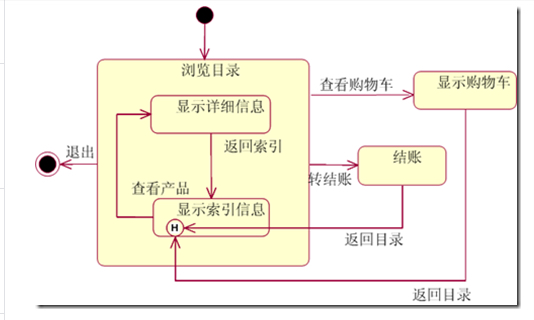


#### 活动与延迟事件

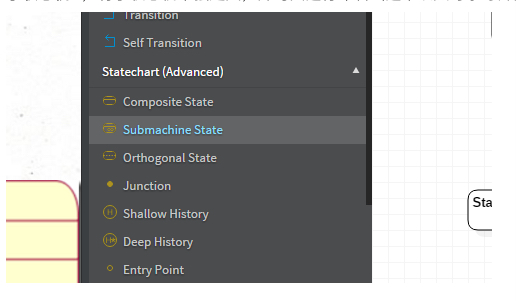


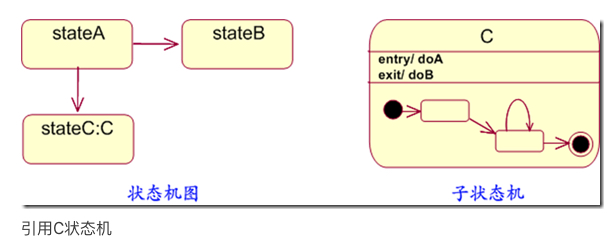
历史  “一个圆圈中加上字母H”，是用来表示历史状态的。

它的含义是：当从状态“结账”和“显示购物车”返回子状态“显示索引信息”时，将进入的是离开时的历史状态。也就是说，转到购物车或结账区之后，再回到“浏览目录”的页面时，其中的内容是不变的，仍然保留原来的信息。



子状态机 ，将子状态机单独定义，并对其进行命名（通常以大写字母开头），然后在需要使用的地方来引用它

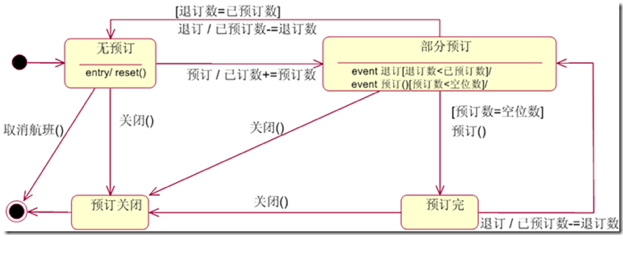






#### 实例

细化状态内的活动与转换

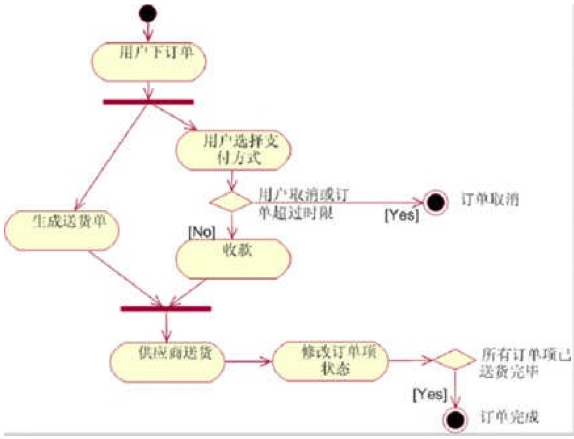


#### 使用复合状态

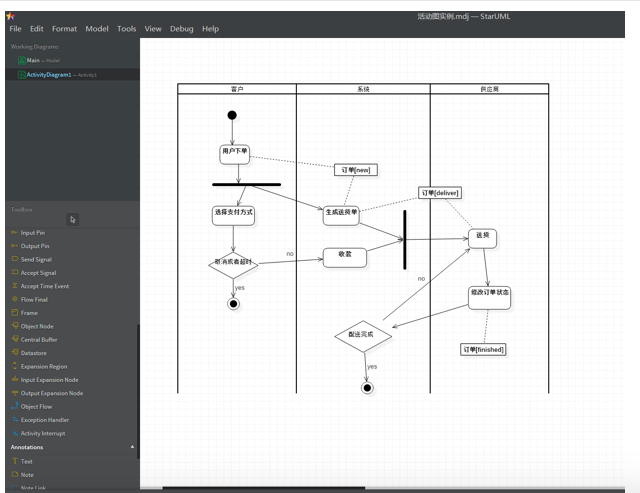


### 活动图(activity diagram)

简单的活动图实例（差不多就是流程图）



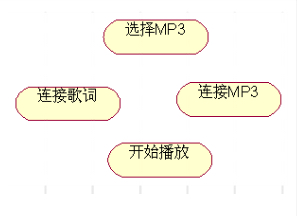
带泳道



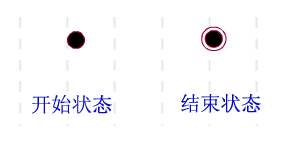
活动图的构成

必要组成元素：

1. 活动：命令的执行，活动的进行。



2、状态：开始状态，结束状态。



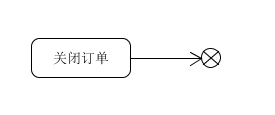
终止节点（Final Node）——分为活动终止节点（activity final nodes）和流程终止节点（flow final nodes）

1. 、活动终止节点表示整个活动的结束



【图形】圆圈+内部实心黑色圆点

1. 、而流程终止节点表示是子流程的结束。



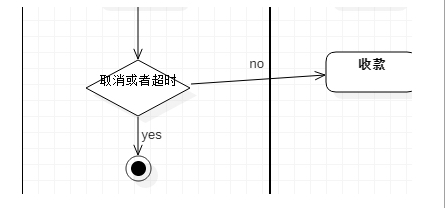
【图形】圆圈+内部十字叉

3、转移：活动之间，活动与状态之间的转换。

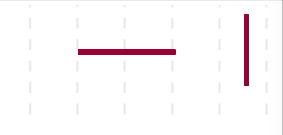


4、判断：对一个动作或者状态进行判断，然后选择要执行的下一步操作。

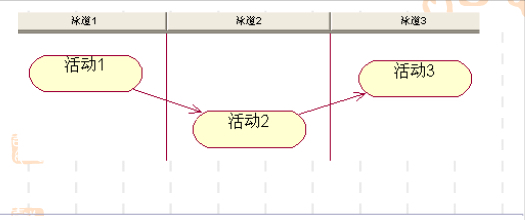
空形菱形表示



1. 同步条：用于并行执行的活动图中。成对出现，并行活动开始和结束都有一跟同步条来表示分歧和汇合。



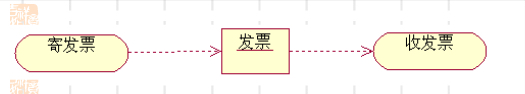
1. 泳道：对活动进行分组。怎么分组—— 每一个区域都代表特定的类，或者是人，或者是部门责任区。



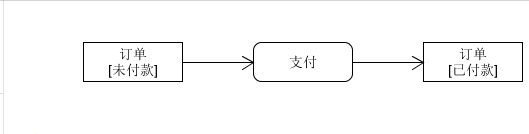
在含有泳道的活动图中，清晰的表明了每个活动的执行对象。在活动图中每个活动只可以属于一个泳道。

7、对象流

活动图中可以将活动涉及到得对象通过依赖将其连接在状态或者活动上。对象用矩形框表示。



状态图中的对象用矩形表示，矩形内是该对象的名称，名称下的方括号表明对象此时的状态。



8、活动图和状态图

活动图和状态图都是状态机的一种表现形式。都是对系统中动态活动进行建模。

两种图的不同：



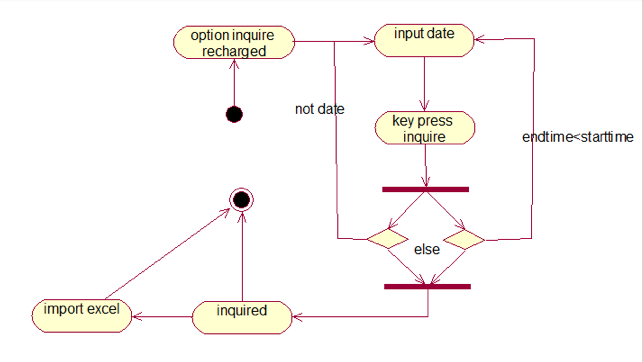
9、活动图和流程图

相似：都是一种流程图。

两种图的不同



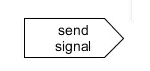
查询充值金额活动图:



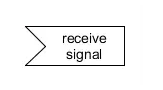
10、时间信号



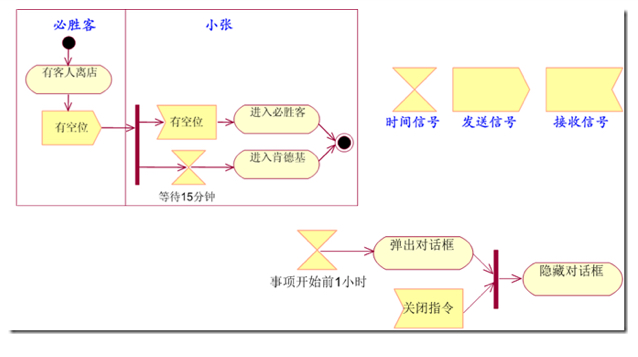
11、发送信号



12、接收信号



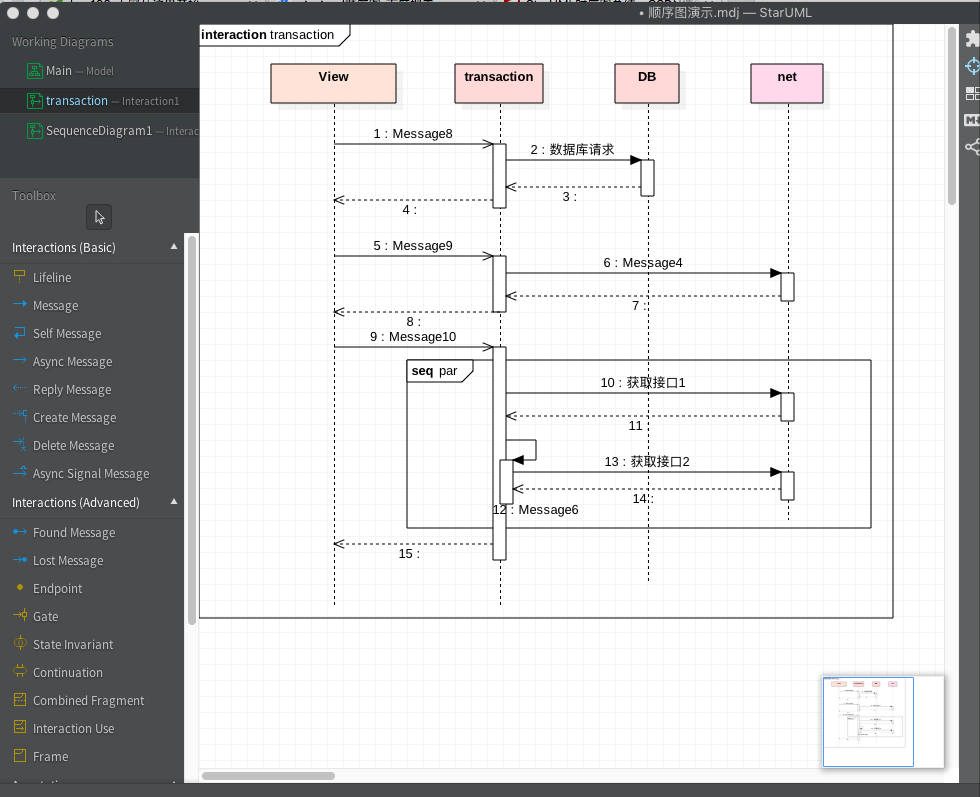
带有发送信号与接收信号的活动图



### 序列图(Sequence Diagram)

顺序图是用来描述对象自身及对象间信息传递顺序的视图。它用来表示用例中的行为顺序。当执行一个用例行为时，顺序图中的每条消息对应了一个类操作或状态机中引起转换的触发事件。它着重显示了参与相互作用的对象和所交换消息的顺序。

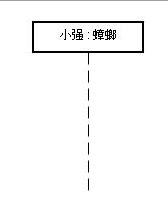
顺序图主要有4个标记符：对象、生命线、消息和激活。



对象是特定行为与属性的集合，表示方式有三种：

1.包括对象名和类名，

如：



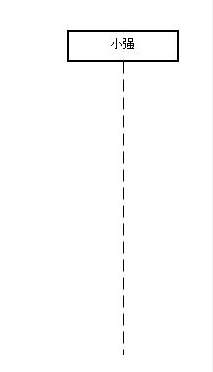
2.只有类名。

如：

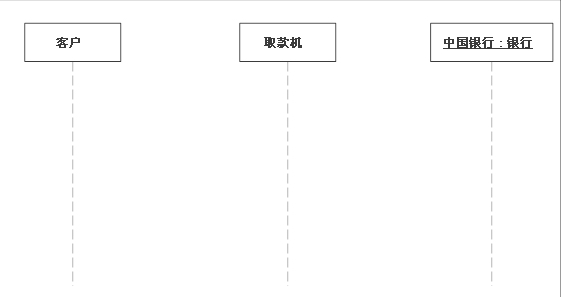


3.只有对象名

如：



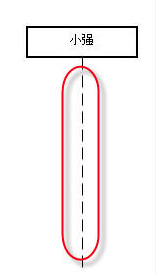
当使用下划线时，意味着序列图中的生命线代表一个类的特定实体。



生命线：

生命线用于描述对象的存在周期，对象下方的虚线就是改对象的生命线。

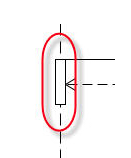
如：



激活：

控制焦点是指活动者或对象处于执行状态的时间段。

如：

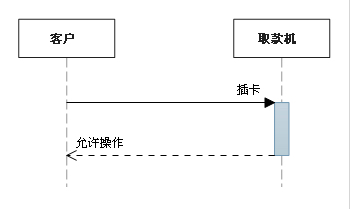


消息：

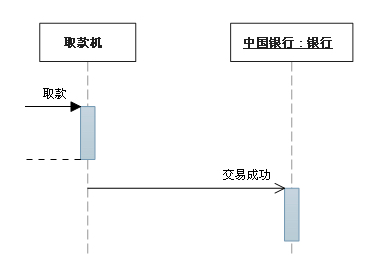
消息用于描述对象间交互的方式及内容。

消息分为四种：同步消息、异步消息、返回消息、自关联消息

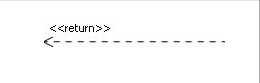
1.同步消息：一个对象向另一个对象发出同步消息后，将处于阻塞状态，一直等到另一个对象的回应。



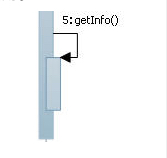
2.异步消息：一个对象向另一个对象发出异步消息后，这个对象可以进行其他的操作，不需要等到另一个对象的响应。



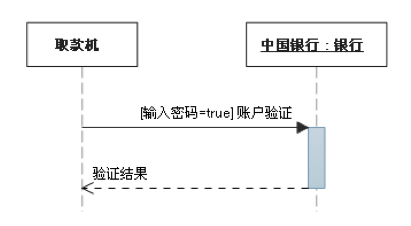
3.返回消息：同步消息的返回消息



4.自关联消息：用来描述对象内部函数的互相调用。



5.约束的符号很简单；格式是: [Boolean Test]



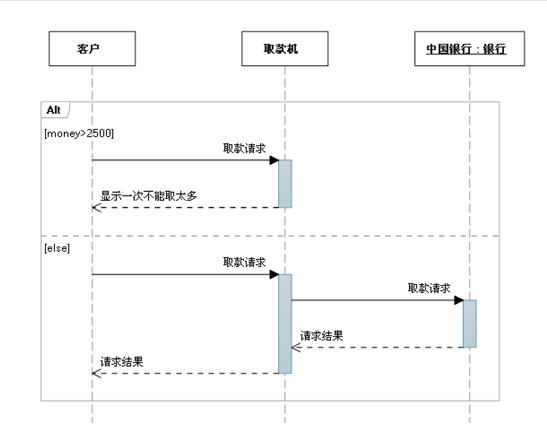
需要说明一下顺序图中对于流程控制的模块：复合片段（Combined Fragments）

复合片段有多种，在此主要介绍一下几种：

条件判断、可选、循环、同步

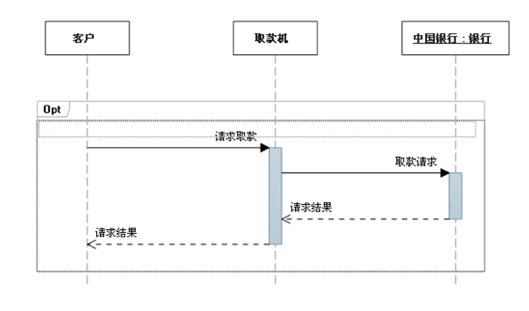
1.条件判断：用于描述代码中if…else…这种结构

标记为“alt”。ps.大概是alternate切换的意思。同alt键。



2.可选：是一种特殊的“条件判断”，它只是一个if，没有else if或else

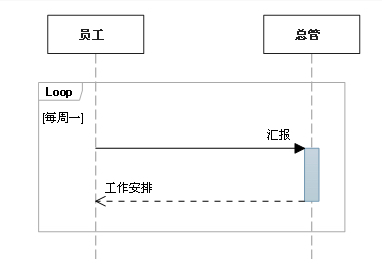
可选的标记为：opt



3.循环：是指代码中的for、while之类的语句块。

循环的标记为：loop

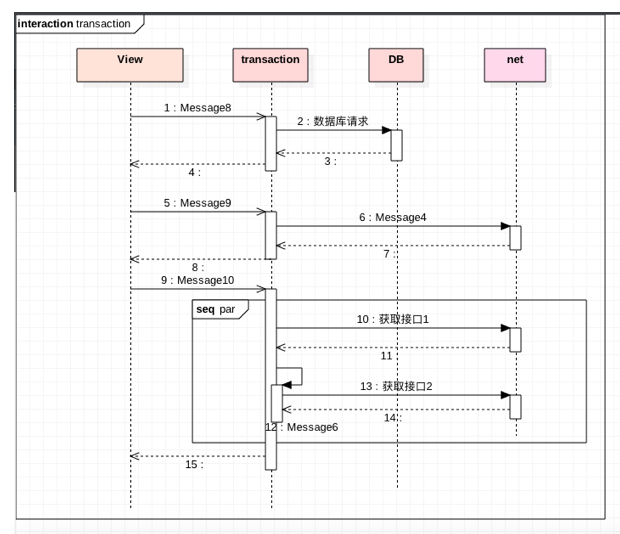
下图中[m,n]是指至少执行m次，最多执行n次。 ps.没看到mn啊。



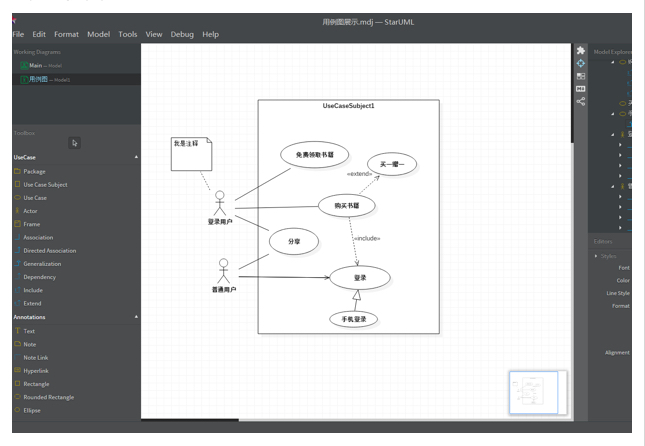
4.同步：用于描述多线程的情况。

同步的标记是：par





### 用例图（Use Case Diagram）



A、用例图 （Use case diagrams ）描述了作为一个外部的观察者的视角对系统的印象。强调这个系统是什么，而不是这个系统怎么工作。用例图描述的是参与者所理解的系统功能，主要元素是用例和参与者，是帮助开发团队以一种可视化的方式理解系统的功能需求。这时处于项目初始，分析用户需求的阶段，不用管怎么实现具体的功能，只要能向客户形象化的表述项目的功能就行。

用例图有四个部分：用例（Use Case), 参与者（Actor),系统边界，关系。

eg.  下图是一个门诊部Make Appointment用例。角色是病人。角色与用例的联系是通讯联系communication association（或简称通讯communication）



角色是人状的图标，用例是一个椭圆，通讯是连接角色和用例的线。

1）参与者（Actor)

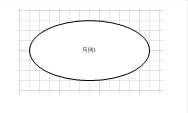
参与者是与系统交互的人或物。首先当然包括我们的开发系统用户，除此之外，与我们开发的系统有关联的其他系统也算是参与者。

在UML图中我们用一个小人表示。

2）用例（Use Case)

用例是参与者可以感受到的系统服务或功能单元。我理解的就是用户可以使用我们开发的项目去做的任何事情

任何用例都不能在缺少参与者的情况下独立存在，同样，任何参与者也必须要有与之关联的用例。在UML图中我们用椭圆表示：

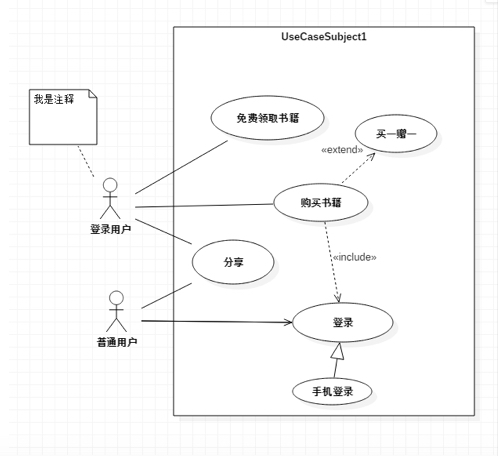


3）系统边界

　　指系统与系统之间的界限。把系统边界以外的同系统相关联的其他部分称为系统环境。在UML图中我们用一个矩形表示。



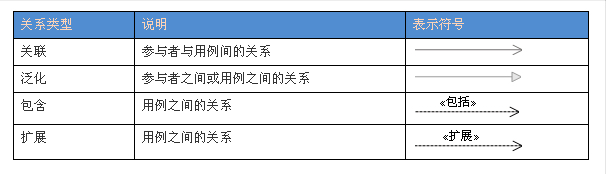
注意一个单独的用例可以有多个角色。



B、关系（Relationship）

常见关系类型有关联、泛化、包含和扩展。

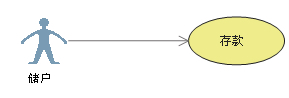
以上各关系在uml图中的表示方式，如下表所示：



a. 关联(Association)

表示参与者与用例之间的通信，任何一方都可发送或接受消息。

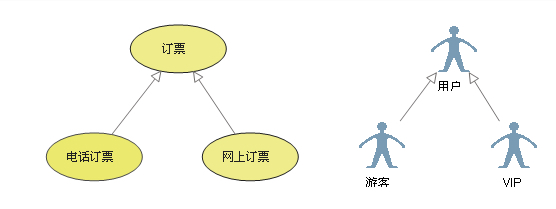
【箭头指向】：指向消息接收方



b. 泛化(Inheritance)

就是通常理解的继承关系，子用例和父用例相似，但表现出更特别的行为；子用例将继承父用例的所有结构、行为和关系。子用例可以使用父用例的一段行为，也可以重载它。父用例通常是抽象的。

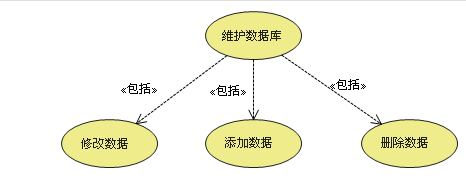
【箭头指向】：指向父用例



c. 包含(Include)

包含关系用来把一个较复杂用例所表示的功能分解成较小的步骤。

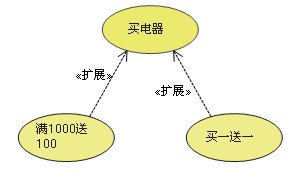
【箭头指向】：指向分解出来的功能用例



d. 扩展(Extend)

扩展关系是指用例功能的延伸，相当于为基础用例提供一个附加功能。

【箭头指向】：指向基础用例



应该如何区分《include》和《extend》 ：

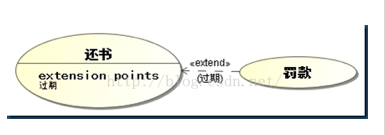
A：《Include》表示一个用例包含另一个用例，即要完成包含用例就一定要执行被包含用例。

B:  《extend》表示一个用例扩展到另一个用例，这里有一点需要注意：在执行一个被扩展用例时，不一定执行扩展用例。即扩展用例的执行是受条件限制的，是可选的。这一点，是区别两个用例之间的关系是《包含》还是《扩展》的依据。

   下面举两个比较直观的例子：



说明：这是一个ATM系统中的两个用例，很显然，在执行“取款”用例的过程必定要执行“银行卡验证”用例。将来在画“取款”的活动图的时候，一定要把“银行卡验证”作为一个步骤加入进去。



C、粒度

用例的粒度指的是用例所包含的系统服务或功能单元的多少。用例的粒度越大，用例包含的功能越多，反之则包含的功能越少。

如果用例的粒度很小，得到的用例数就会太多。反之，如果用例的粒度很大，那么得到的用例数就会很少。

如果用例数目过多会造成用例模型过大和引入设计困难大大提高。 如果用例数目过少会造成用例的粒度太大，不便于进一步的充分分析。

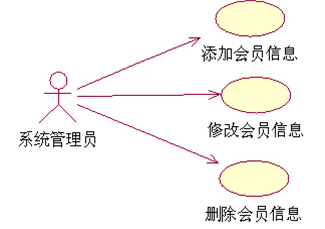
比如：网站后台管理系统中的会员信息维护用例，管理员需要进行添加会员信息、修改会员信息、删除会员信息等操作。

这里写图片描述



我们还可以根据具体的操作把它抽象成3个用例，它展示的系统需求和单个用例是完全一样的。

这里写图片描述

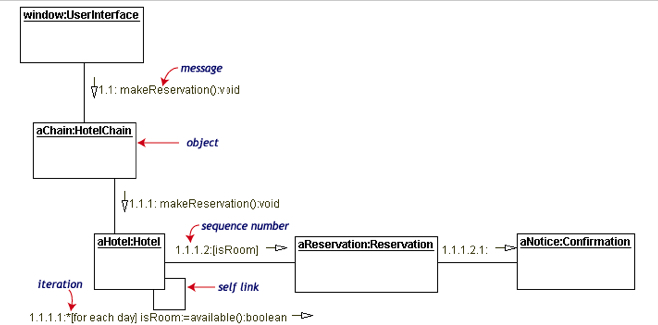


### 协作图 （\*）

协作图也是互动的图表。他们像序列图一样也传递相同的信息，但他们不关心什么时候消息被传递，只关心对象的角色。在序列图中，对象的角色放在上面而消息则是连接线。

对象角色矩形上标有类或对象名（或者都有）。类名前面有个冒号（：）。

协作图的每个消息都有一个序列号。顶层消息的数字是1。同一个等级的消息（也就是同一个调用中的消息）有同样的数字前缀，再根据他们出现的顺序增加一个后缀1，2等等。

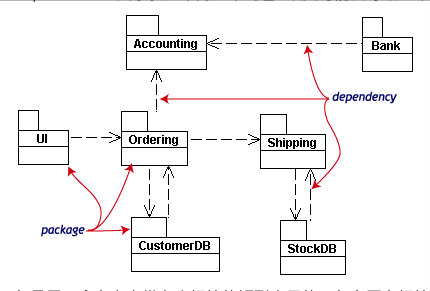


### 包图（\*）

为了简单地表示出复杂的类图，可以把类组合成包packages。一个包是UML上有逻辑关系的元件的集合。下面这个图是是一个把类组合成包的一个商业模型。

dependencies关系。如果另一个的包B改变可能会导致一个包A改变，则包A依赖包B。

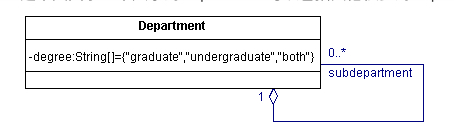
包是用一个在上方带有小标签的矩形表示的。包名写在标签上或者在矩形里面。点化线箭头表示依赖



### 对象图（\*）

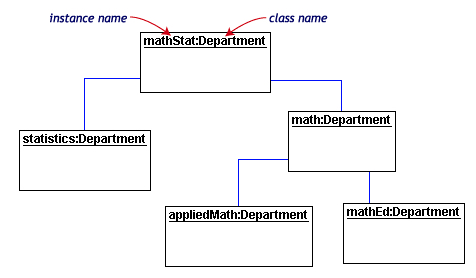
对象图Object diagrams用来表示类的实例。他们在解释复杂关系的细小问题时（特别是递归关系时）很有用。

这个类图示一个大学的Department可以包括其他很多的Departments。



这个对象图示上面类图的实例。用了很多具体的例子。

UML中实例名带有下划线。只要意思清楚，类或实例名可以在对象图中被省略。



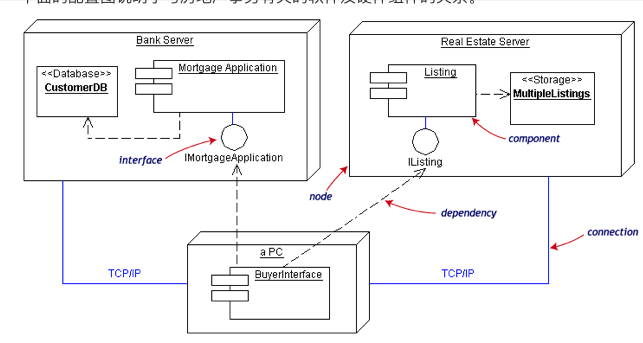
每个类图的矩形对应了一个单独的实例。实例名称中所强调的UML图表。类或实例的名称可能是省略对象图表只要图的意义仍然是明确的。

### 9.组件与配置图（\*）

组件component是代码模块。组件图是是类图的物理实现。

配置图Deployment diagrams则是显示软件及硬件的配置。

下面的配置图说明了与房地产事务有关的软件及硬件组件的关系。



物理上的硬件使用节点nodes表示。每个组件属于一个节点。组件用左上角带有两个小矩形的矩形表示。

## 总结

startuml适合画专业的uml图形，StarUML是一种生成类图和其他类型的UML图表的工具。

参考文献：

1、UML类图几种关系的总结

http://www.uml.org.cn/oobject/201609062.asp

2、使用StarUML画类图

https://blog.csdn.net/dean\_deng/article/details/44657755

3、UML 及 StarUml

https://segmentfault.com/a/1190000011556007

4、staruml反向(逆向)Java工程通过代码生成类图

https://jingyan.baidu.com/article/414eccf615f2c46b431f0a87.html

5、用例图中，应该如何区分《include》和《extend》https://blog.csdn.net/xiao190128/article/details/50808770

6、浅谈UML中常用的几种图——用例图

https://www.cnblogs.com/vathe/p/7349816.html

7、用例图详解

https://blog.csdn.net/mj\_ww/article/details/53020080

8、UML学习笔记（五）--顺序图

https://blog.csdn.net/qianmodanshang/article/details/53183540

9、StarUML时序图总结

https://blog.csdn.net/achuo/article/details/47448313

10、UML学习笔记（四）--活动图

https://blog.csdn.net/qianmodanshang/article/details/53183436

11、UML 顺序图

https://www.cnblogs.com/yxx123/p/5227272.html

12、UML——状态图

https://www.cnblogs.com/sura/archive/2012/07/01/2572083.html

13、UML建模之状态图（Statechart Diagram）

https://blog.csdn.net/heshengfen123/article/details/9361959

14、UML建模——活动图（Activity Diagram）

http://www.cnblogs.com/xiaolongbao-lzh/p/4591953.html

15、UML图之四——活动图

https://blog.csdn.net/wangyongxia921/article/details/8250103